THE ISLAMIC UNIVERSITY – FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

LCOM 3010 – Introduction to Computer Architecture Lab

Lab #6
CONTROL UNIT DESIGN

BY

Eng: Rasha Ghazy Sammour

Submitted for

Dr. Mohammed Mikki

March 31, 2021

Control Unit Design

Objective:

1. Show the basic top level functionality, organization and architecture of a computer.
2. Designing a single instruction CPU which contains a controller.

Theory: Design of Control Unit:

The control logic depends on the details of the devices in the control path, and on the individual bits in the op code for the instructions. The arithmetic and logic operations for the r-type instructions also depend on the `funct` field of the instruction.

The datapath elements we have used are:

- a 32 bit ALU with an output indicating if the result is zero
- adders
- MUX's (2 line to 1-line)
- a 32 register $\times$ 32 bits/register register file
- individual 32 bit registers
- a sign extender
- instruction memory
- data memory

We will design the control logic to implement the following instructions (others can be added similarly):

Name	Op-code					
	Op5	Op4	Op3	Op2	Op1	Op0
R-format	0	0	0	0	0	0
`lw`	1	0	0	0	1	1
`sw`	1	0	1	0	1	1
`beq`	0	0	0	1	0	0
`j`	0	0	0	0	1	0

Note that we have omitted the immediate arithmetic and logic functions.

The `funct` field will also have to be decoded to produce the required control signals for the ALU.

The control signals

The signals required to control the datapath are the following:

- `Jump` — set to 1 for a `jump` instruction
- `Branch` — set to 1 for a `branch` instruction
- `MemtoReg` — set to 1 for a `load` instruction
- `ALUSrc` — set to 0 for r-type instructions, and 1 for instructions using immediate data in the ALU (`beq` requires this set to 0)
- `RegDst` — set to 1 for r-type instructions, and 0 for immediate instructions
- `MemRead` — set to 1 for a `load` instruction
- `MemWrite` — set to 1 for a `store` instruction
- `RegWrite` — set to 1 for any instruction writing to a register
- `ALUOp` (k bits) — encodes ALU operations except for r-type operations, which are encoded by the `funct` field

For the instructions we are implementing, `ALUOp` can be encoded using 2 bits as follows:

`ALUOp[1]`	`ALUOp[0]`	Instruction
0	0	memory operations (`load, store`)
0	1	`beq`
1	0	r-type operations

The following tables show the required values for the control signals as a function of the instruction op codes:

Instruction	Op-code	RegDst	ALUSrc	MemtoReg	Reg Write
r-type	0 0 0 0 0 0	1	0	0	1
`lw`	1 0 0 0 1 1	0	1	1	1
`sw`	1 0 1 0 1 1	x	1	x	0
`beq`	0 0 0 1 0 0	x	0	x	0
`j`	0 0 0 0 1 0	x	x	x	0

Instruction	Op-code	Mem Read	Mem Write	Branch	ALUOp[1:0]	Jump
r-type	0 0 0 0 0 0	0	0	0	1 0	0
`lw`	1 0 0 0 1 1	1	0	0	0 0	0
`sw`	1 0 1 0 1 1	0	1	0	0 0	0
`beq`	0 0 0 1 0 0	0	0	1	0 1	0
`j`	0 0 0 0 1 0	0	0	0	x x	1

This is all that is required to implement the control signals; each control signal can be expressed as a function of the op-code bits.

For example,

$\text{RegDst} = \overline{\text{Op5}} \cdot \overline{\text{Op4}} \cdot \overline{\text{Op3}} \cdot \overline{\text{Op2}} \cdot \overline{\text{Op1}} \cdot \overline{\text{Op0}}$

$\text{ALUSrc} = \text{Op5} \cdot \overline{\text{Op4}} \cdot \overline{\text{Op2}} \cdot \text{Op1} \cdot \text{Op0}$

All that remains is to design the control for the ALU.

The ALU control

The inputs to the ALU control are the `ALUOp` control signals, and the 6 bit `funct` field.

The `funct` field determines the ALU operations for the r-type operations, and `ALUOp` signals determine the ALU operations for the other types of instructions.

Previously, we saw that if `ALUOp[1]` was 1, it indicated an r-type operation. `ALUOp[0]` was set to 0 for memory operations (requiring the ALU to perform an `add` operation to calculate the address for data) and to 1 for the `beq` operation, requiring a subtraction to compare the two operands.

The ALU itself requires three inputs.

The following table shows the required inputs and outputs for the instructions using the ALU:

Instruction	ALUOp	`funct`	ALU operation	ALU control input
`lw`	0 0	x x x x x x	add	0 1 0
`sw`	0 0	x x x x x x	add	0 1 0
`beq`	0 1	x x x x x x	subtract	1 1 0
`add`	1 0	1 0 0 0 0 0	add	0 1 0
`sub`	1 0	1 0 0 0 1 0	subtract	1 1 0
`and`	1 0	1 0 0 1 0 0	AND	0 0 0
`or`	1 0	1 0 0 1 0 1	OR	0 0 1
`slt`	1 0	1 0 1 0 1 0	set on less than	1 1 1

Extending the instruction set

What is necessary to add another instruction to the instruction set?

First, the appropriate elements must be added to the datapath.

Second, any control elements must be added, and appropriate control signals identified.

Third, the control logic must be extended to enable the appropriate elements in the datapath.

Let us consider adding the instruction `or immediate` (`ori`)
It has the form

```
ori $s1, $s2, imm
```

Its function is to perform the logical OR of the contents of register `$s2` with the *zero extended* immediate data field `imm`, storing the result in register `$s1`.

```
$s1 ← $s2 | ZeroExtend[imm]
```

It has op-code `0 0 1 1 0 1` and is an immediate type instruction.

First — add elements to the data path

Examining the data path, the ALU can perform the OR operation, but the extender unit only supports sign extension. It can be replaced by a unit, `sign or zero extend`, which can perform both functions.

Second — add control elements

This new unit requires a new control signal to select the zero extend function (0) or the sign extend function (1).
We will label the new signal `ExtOp`.

Also, the 2-bit control signal `ALUOp` only encodes the operations `add` and `subtract`. Adding a third bit would allow the encoding of the operations `AND` and `OR`.
It can be encoded as follows:

`ALUOp[2]`	`ALUOp[1]`	`ALUOp[0]`	Instruction
0	0	0	memory operations (`load, store`)
0	0	1	`beq` (subtract, in the ALU)
0	1	0	`ori`
1	x	x	r-type operations

The following diagram shows the changes required to the datapath:

Third - the control logic

The truth table for the ALU control unit extends to:

Instruction	ALUOp	`funct`	ALU operation	ALU control input
`lw`	0 0 0	x x x x x x	add	0 1 0
`sw`	0 0 0	x x x x x x	add	0 1 0
`beq`	0 0 1	x x x x x x	subtract	1 1 0
`ori`	0 1 0	x x x x x x	OR	0 0 1
`add`	1 0 0	1 0 0 0 0 0	add	0 1 0
`sub`	1 0 0	1 0 0 0 1 0	subtract	1 1 0
`and`	1 0 0	1 0 0 1 0 0	AND	0 0 0
`or`	1 0 0	1 0 0 1 0 1	OR	0 0 1
`slt`	1 0 0	1 0 1 0 1 0	set on less than	1 1 1

For the `ori` instruction, the following settings are required for the remaining control signals:

Jump	0	
Branch	0	
MemRead	0	
MemWrite	0	
MemtoReg	0	
ALUSrc	1	ALU operand is from the extender
RegDst	0	`rt` is the destination register
RegWrite	1	result will be written in `reg[rt]`
ExtOp	0	zero extend

The modified tables for the control signals are:

Inst.	Op-code	RegDst	ALUSrc	MemtoReg	Reg Write
r-type	0 0 0 0 0 0	1	0	0	1
`lw`	1 0 0 0 1 1	0	1	1	1
`sw`	1 0 1 0 1 1	x	1	x	0
`beq`	0 0 0 1 0 0	x	0	x	0
`j`	0 0 0 0 1 0	x	x	x	0
`ori`	0 0 1 1 0 1	0	1	0	1

Inst.	Op-code	Mem Read	Mem Write	Branch	Jump	ALUOp 2 1 0	ExtOp
r-type	0 0 0 0 0 0	0	0	0	0	1 0 0	x
`lw`	1 0 0 0 1 1	1	0	0	0	0 0 0	1
`sw`	1 0 1 0 1 1	0	1	0	0	0 0 0	1
`beq`	0 0 0 1 0 0	0	0	1	0	0 0 1	1
`j`	0 0 0 0 1 0	0	0	0	1	x x x	x
`ori`	0 0 1 1 0 1	0	0	0	0	0 1 0	0

Some of the control logic may have to be modified. For example, the logic generating the signal `ALUSrc` would have to ensure that the value 1 was set for the `ori` instruction:

$$\text{ALUSrc} = \text{Op5}\cdot\overline{\text{Op4}}\cdot\overline{\text{Op2}}\cdot\text{Op1}\cdot\text{Op0} + \overline{\text{Op5}}\cdot\overline{\text{Op4}}\cdot\text{Op3}\cdot\text{Op2}\cdot\overline{\text{Op1}}\cdot\text{Op0}$$

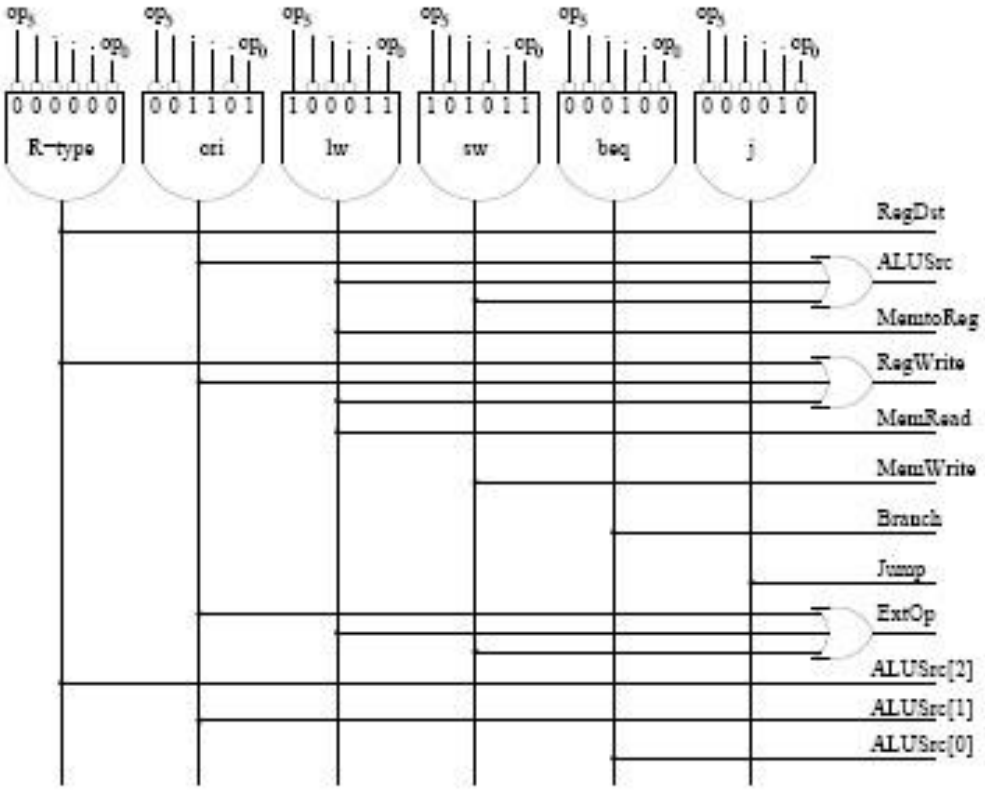
The new control signal, `ExtOp` can be evaluated as:

$$\text{ExtOp} = \text{Op5}\cdot\overline{\text{Op4}}\cdot\overline{\text{Op2}}\cdot\text{Op1}\cdot\text{Op0} + \overline{\text{Op5}}\cdot\overline{\text{Op4}}\cdot\overline{\text{Op3}}\cdot\text{Op2}\cdot\overline{\text{Op1}}\cdot\overline{\text{Op0}}$$

Other control logic implementations

Because there are only a few instructions to be implemented, the control logic for this processor implementation is probably best implemented using simple logic functions as shown previously.

It is quite common to implement simple controllers as a PLA. Following is a PLA implementation for the processor we have designed so far:



Assignment Statements:

- Add NOR instruction to instruction set on control unit. By using Logisim.